RSRE

MEMORANDUM No. 4188

ROYAL SIGNALS & RADAR ESTABLISHMENT

ON THE DESIGN AND IMPLEMENTATION OF A SECURE COMPUTER SYSTEM

Authors: P F Torry & S R Wiseman

PROCUREMENT EXECUTIVE,
MINISTRY OF DEFENCE,
R S R E MALVERN,
WORCS.

ROYAL SIGNALS AND RADAR ESTABLISHMENT

Memorandum 4188

Title : On the Design and Implementation of a Secure Computer System

Author : Phil F.Terry[†] and Simon R. Wiseman

Date : June 1988

Summary

This memorandum presents the results of a one year contract (A94c/2711), carried out by TSL Communications Ltd., which considered RSRE's SMITE secure computer system. The contract provided a peer review of the architecture proposals, characterised the essential architectural elements and formulated the security oriented top level model.

[†] TSL Communications Ltd
20 Alan Turing Road
Surrey Research Park
Guildford,Surrey, GU2 5YF

ABSTRACT

SMITE is a novel computer architecture implementing a new security policy model which is proposed for use in Government and Military environments where high assurance of complex confidentialty and integrity based security policies is required.

This report records the results of a one year contract (A94c/2711) carried out by TSL Communications Ltd. This provided a peer review of the proposed architecture, characterised the essential architectural elements and formulated the security oriented top level model. In this way it provided a baseline definition of SMITE to aid the future way forward for the project.

Keywords: Great Britain, Multiprocessor, [illegible]

CONTENTS

1. INTRODUCTION

1.1. Project Background

SMITE has resulted from a program of research into Multi-Level Computer Security begun by RSRE over a decade ago. At that time RSRE were working on a number of simple separation approaches to security with projects involving industry and academia such as,

> Secure User Environment (SUE), out of which came the Distributed Secure System (DSS) development and Rushby's Proof of Separability approach to verification of separation kernels,
>
> Secure Network Interface Protocol (SNIP), which involved investigation of the problems of plaintext bypass in an encryption device where headers for protocols must be passed unencrypted,
>
> and finally, Secure Packet Front End (SPFE), investigating the security requirements for packet switching store and forward systems.

Just prior to this work the USA had launched its Computer Security Initiative, involving the setting up of the National Computer Security Center to coordinate the industrial and academic efforts into computer security and to widely disseminate the requirements for security. The impetus for this initiative was a result of a study into the problem which culminated in the Anderson Panel Report.

RSRE issued a contract for a study to produce recommendations for the best use of effort towards similarly widening the knowledge of the issues and requirements for computer security in the UK.

This contract, awarded to Plessey, resulted in a comprehensive report on computer security in computer networks [Andrews81]. This report made extensive use of the experience gained in the USA and provides a comprehensive review and discussion of the problems and technological solutions involved in Multi-level Computer Security. The reader requiring such background is strongly recommended to review this report, which despite its age is still relevant.

The Plessey report recognised that the then current RSRE projects formed a low-level application of existing techniques to existing problems and thus formed an approach which we might with hindsight characterise as a "nuts and bolts" approach. As a way forward the report recommended the immediate deployment of effort to produce a "mid-range" Trusted Computing Base (TCB) system for limited functionality network devices that would be required in the near future. In the background a longer term research effort should be mounted to produce highly secure general purpose computer systems.

The report identified three phases for this Secure Communications Processor project. Phase one, or SCP1, was a collective term for the low level approaches such as SUE and SNIP etc, SCP2 was the development of the mid-range TCB based on current hardware and software developments, and finally SCP3, the general purpose computer system, for which specialised hardware and software techniques could be applied [Barnes85].

SMITE is in fact the name of the third phase, SCP3, project [Wiseman86a, Wiseman86b]. The name has been changed in recognition of the fact that SMITE is a general purpose computer system and is not limited to network component functionality as the term SCP3 might imply.

1.2. Technical Background

The SMITE proposals for secure system development have been influenced from a number of sources which will be referenced throughout this document.

Our approach to policy issues has been profoundly influenced by the many useful discussions and workshops which arose from the Clark-Wilson paper, "A Comparison of Commercial and Military Security Policies" [Clark87]. The manner in which this influences the SMITE approach is discussed fully in the next "Overview" section, but briefly, the concept of the requirement for consistency between the internal system state and the external environment, presented in their model as the Separation of Duty concept, is shown to be the fundamental security mechanism of not only integrity but confidentiality and assurance in general.

Our modelling approach is basically that expounded in the seminal "Mathematical Foundations" paper of Bell-LaPadula [Bell73a], however our model, the notions of confidentiality, the axioms, and techniques used are profoundly different from those found in the later papers [Bell73b,Bell74 and Bell76]. Our approach has been influenced from a number of sources such as Bell-LaPadula, McLean [McLean87] and various information flow techniques such as Non-Interference [Goguen82] and Separability [Rushby81].

The model is sufficiently powerful to express such notions, as well as those of integrity exemplified by Biba [Biba77], Clark and Wilson's well formed transactions and their formal notion of Separation of Duty within a consistent framework.

Our approach to assurance issues of security polices, system designs and implementations draws from discussions at workshops of the DTI's Commercial Computer Security Centre and at a UK interest group on Integrity policies formed because of interest in the Clark-Wilson paper.

The architecture and software development approach proposed is a direct development of work by RSRE's CC2 division and is a capability architecture supporting a first class procedure regime. [Foster82a, Foster82b, Foster83, Currie82, Currie85]

The existence of a team with the experience and inspiration to draw these many disparate elements together into the SMITE proposals is entirely due to the foresight and perseverance of RSRE's CC1 division in maintaining the SCP research program, which has produced convincing demonstrations of research results such as SCP2 and DSS.

1.3 Structure of the Remainder of the Document

Section 2, Overview, is a fairly expansive, informal presentation of the overall SMITE approach to security. It develops concepts of security from scratch and relates these concepts to existing work and practice in an entirely informal manner. The section therefore has a tutorial appearance giving background material. Usually, after having described a section as tutorial, the experienced reader is encouraged to skip the section and proceed directly with the "meat" of the subject. In this case we strongly advise against this approach for the following reason.

The flavour of the SMITE approach to security is that it is a coherent, balanced approach to the whole problem of security but with no loss of rigour in the approaches to individual aspects of security such as confidentiality, integrity or assurance. Ultimately, it is our belief that the presentation of the SMITE approach will satisfy all partisan readers that their area has not been sacrificed for the sake of consensus.

In order to argue this in a document aimed at a general readership, it is necessary that readers approach the arguments presented with their preconceptions and connotations of words and phrases temporarily laid aside and use only the definitions and concepts presented within this document. It is in order to develop this frame of mind in the reader that Section 2 has been provided. It is only when the complete presentation of the approach has been thus read, and hopefully digested, that the readers should return to their critical approach. By this means it is hoped that any criticism of the approach is of a fundamental nature rather than the usual annoying misunderstanding of terminology.

Section 3, Security Policy Model, presents the formal policy modelling approach followed by a model and axioms which capture the concepts developed in the Overview. The basic execution nature of the model is shown to consist of a small number of basic transitions which can be embodied in a relatively small number of transition rules. Allthough the model elements and transitions are quite simple it is shown that it is sufficient to model complex applications and general purpose computer systems.

Section 4, The SMITE Architecture, describes the underlying architectural features which are required to implement the policy model. It then goes on to describe the intended architecture for use in the SMITE project and describes the correspondence between the architecture and the policy model in an informal manner. In an operational development the correspondence would be formally demonstrated by continuing the refinement of the model to design, to implementation. The architecture proposed is believed to be the best available for implementing the model and the informal presentation is actually felt to be sufficient for the purposes of providing firm evidence that the model can be implemented to high assurance.

Section 5, Future Plans, discusses future research and development which could usefully be carried out to progress both the SMITE project and the application of the modelling approach to a wider range of system and problems.

2. OVERVIEW

Because of the radical nature of the SMITE approach this section will attempt to develop from scratch the notions of security which we hold as fundamental, using only the intuitive notions in wide circulation. Our test of acceptability of a notion, or defintion of a term, is the legal one of "the view of the man on the Clapham Omnibus", represented by the English Dictionary.

In doing this we will relate these issues to existing work by presenting interpretations of such works in the terms which we develop. While this leads to a rather tutorial appearance it has the advantage that it allows us to discuss our relationship to existing work without misunderstandings due to unstated assumptions and connotations which experienced readers may bring with them. Such readers may still disagree with our conclusions but the disagreements will at least be on fundamental issues rather than details of the presentation.

2.1 What is Security?

SMITE is an approach to producing secure information systems. This leads us to ask what is security? A standard dictionary definition is as follows.

> **security** n. **1** the state of being secure. **2** assured freedom from poverty or want.... Archaic carelessness or overconfidence
>
> **secure** adj. **1** free from danger, damage etc. **2** free from fear, care etc..... Archaic careless or overconfident. [From Latin securus free from care, from se- without + cura, care]

This gives us the typical manufacturers view of a customer defintion of security, "I want a system that I don't care about because I am assured that it is free from all the etc's which might assail me in the future but which I can't enumerate at the moment".

The archaic definitions of carelessness and overconfidence seem to shout out from the past the dire consequences of such an approach and echo the concerns of the modern prophets of doom, "No security is better than illusory security".

A sophisticated view of security in these terms can however be described as assurance of freedom from fear of specified attacks against specified elements of a system. To define security in these terms is a tripartite affair,

i. specify which properties of which elements of the system are important,

ii. specify what protection from which attacks a threat analysis requires,

iii. specify how much assurance is required that such defences are successful.

Taken in its general English sense this definition of security is not at odds with either military or commercial security practice.

2.1.1 MILITARY SECURITY

The military and government arena has in the past taken a more defintive approach to security and attempted to codify and lay down standards for secure systems. An impartial view of these attempts shows that quite naturally they have emphasised a particular property, confidentiality, as paramount to security.

In codifying these concepts the military have produced many formal definitions, or models, of confidentiality. A dictionary analysis of this term will suffice for our discussions here and is as follows.

Confidential adj. **1** spoken, written, or given in confidence; secret; private. **2** *entrusted with anothers confidence or secret affairs.*

Confidence n. **1** feeling of trust in a person or thing. ..... **4** something *confided or entrusted.* **5 in confidence** as a secret.

Secret adj. **1** kept hidden or separate from the knowledge of others. **4** able or tending to keep things private or to oneself.

private adj. **1** not widely known; confidential; secret. [*from Latin privatus* belonging to one individual]

Confidentiality from the above seems to be about knowledge of things and its distribution amongst individuals.

The aspect of confidentiality that things are kept secret or private is an easily stated and implemented requirement for the discrete identification of things and ensuring that they are attributed to only one individual, an isolation policy.

policy *n.* **1** *a plan of action adopted or pursued by an individual, government, party, business, etc.*

The assurance of such a policy can be very high because it is basically saying that if you want something done in such a way that no one else knows about it - do it yourself.

The notion that distribution of things amongst individuals is allowed "in confidence" in the definition recognises that in reality this isolation policy is not practical and that one is forced to delegate tasks. This problem of delegating or sharing a task with others leads to a quite natural extension of isolation policy as follows. If things are isolated and are secret, that is private, known only to one individual, then if an individual can establish a basis of trust in another individual he may pass knowledge of a secret thing to that other individual.

In this light a generic confidentiality policy requires that things are isolated and attributable to individuals, lays down the means by which a basis of trust in individuals is established, *ie* that they don't pass on or leak secrets given to them, and requires that things are not delegated to multiple individuals other than as permitted by properly established trust.

2.1.2 COMMERCIAL SECURITY

Some have argued that impartial analysis of commerical security shows it to be biased to a particular property, integrity.

integrity n. **1** adherence to moral principles; honesty. **2** the quality of being unimpaired; soundness. **3** unity; wholeness(see INTEGER).

integer n. .....**2** an individual entity or whole unit [from the Latin untouched, entire from tangere, to touch]

honest adj. **1** not given to lying, cheating, stealing etc.; trustworthy.

sound[2] adj. **1** free from damage, injury, decay etc..... **5** valid, logical or justifiable.

Integrity from the above, and as it is normally used in the context of security, is the notion that things have "desirable" properties such as that they are undamaged, valid, unimpaired etc.

As with the simple secrecy or privacy notion of confidentiality addressing this simple requirement can again be fulfilled with high assurance by a policy of "do it yourself if you want it done right". In the same way therefore it is not surprising that in reality this is not practical and we require a basis of trust for delegating tasks.

Therefore a generic integrity policy requires that things are discretely identifiable and attributed to an individual, lays down a basis for establishing trust of individuals ie they don't damage it, render it invalid etc, and requires that things are not delegated to multiple individuals other than as permitted by properly established trust.

2.1.3 ESTABLISHING TRUST IN INDIVIDUALS

For both confidentiality and integrity, once we are beyond the "do it yourself" syndrome, we are therefore concerned with establishing trust in individuals. Thus individuals are "studied" or "investigated" to see if there are any past behavioural traits of, or future predispositions to, engaging in activities which are prejudicial to the properties of the tasks or information we are delegating to them. Once this is done we can issue a certificate to individuals, and appraise them of the properties or procedures which they must uphold when handling the delegated task or information. For confidentiality individuals therefore are required to label data correctly and only give it to individuals who are cleared and have a need to know. For integrity individuals must not defraud the company or allow others to do so.

In this way we are depending on the cooperation of all delegated individuals to uphold the common aim of the policy, that is we assume they all share the same objectives. Now while this is a reasonable assumption it only provides us with assurance that our policy is being complied with to the level established by our behavioural checks on individuals. Recognising that these checks are fallible, that an individuals motivation to uphold the policy may change between checks (assuming they are periodic!), and that in a complex task being cooperatively carried out by many individuals mistakes and oversights can occur, it is reasonable that we seek additional assurance by arranging that the collective system used by the individuals to achieve their common delegated task makes it as difficult as possible to violate the policy. That is we seek to control the actions of the individuals.

Obviously, if we control every action of every individual we are actually doing the job ourselves which we know is secure but impractical. Therefore we must leave individuals a degree of discretion in their actions but seek to limit the damage caused by an errant or subverted individual.

Together, this "loose" control of individuals behaviour plus an a priori means of labelling the degree to which individuals share objectives for the remaining uncontrolled behaviour is the basis of trust which allows us to delegate tasks to others.

The "them" and "us" of these and the following discussions should not be taken to mean peer users of the system. The view we are taking is the wider view that the Government, State, Nation, or Company delegates tasks to its citizens, inhabitants, or employees etc but requires that the interests of the corporate body is upheld. In the case of the government and its citizens those whose interests are being protected and those who might subvert them are one and the same population, in the case of a company it is the shareholders whose interests are protected from management and employees, which may be a disjoint population.

Individuals "Motivation"

For commercial or government sensitive but unclassified arenas a notion of security enforcement based on individual background checks to establish an individual's loyalty may be considered unacceptable because of concerns for individual privacy and civil rights issues. Clark-Wilson have shown however, that in the commercial sector this same goal is achieved in practice without background checks because of implicit assumptions which can be formed about an individuals "motivation".

The example that springs to mind is that a highly paid, career oriented manager of a Bank is unlikely to collude with lowly paid counter clerks in a petty fraud. Thus if transactions by the clerks require a final counter signature by the manager, who is known to make random spot checks on the veracity of the transactions, fraud by counter clerks is inhibited. Major fraud by the manager alone is prevented by the Bank organisation which does not allow highly placed individuals to carry out the basic transactions, only to countersign and check them. Major fraud by the manager in collusion with his staff is also prohibited by the paradoxical motivation of the clerks not to cooperate, "The big fish never go to prison, its always the little guy who carries the can".

This notion of separation of duties amongst individuals who, by background check, assumptions about motivations, or some combination of the two, can be assured of carrying out a task without collusion to violate policy, is only applicable to ensuring the integrity of tasks. Confidentiality cannot be ensured by this means. For confidentiality it only takes one subversive or accidental action by one individual to disclose a secret thus the more individuals that acquire knowledge of it the greater the chance of disclosure. Conversely for integrity, individuals able to complete the whole task may be subverted or careless thus by ensuring that many individuals must cooperate in a highly coordinated manner the chance of undesirable results is reduced.

Because of this divergence of properties it may not seem clear how these two aspects, confidentiality and integrity, are coordinated into a coherent notion of security.

2.1.4 TOTAL SECURITY

The most obvious relationship of these two mechanisms is that while the only reason for wanting a task of information processing to be carried out at all presupposes that it will be done correctly, we require it to be carried out in confidence. In other words confidentiality control supercedes integrity control.

In a system of confidentiality controls, if we are to avoid "doing it ourselves" but at the same time retain assurance that confidentiality is upheld, we are forced to implement the controls so as to make the worst case assumption of the behaviour of individuals in terms of the sensitivity labels of information they produce from initially labelled information. For this reason there is a constant need to "trim" overclassified, generated information. If the system in its application role within an organisation is taken into account, there also arises needs to change the sensitivity labels of information in the machine to reflect the changed perceptions of the environment. For these reasons there is a requirement that individuals can change the control mechanisms that are implemented to effect controls on individuals. There is obviously a requirement for maintaining the integrity of these controls in the sense that an inappropriate or fraudulent change is not made. In other words in practice integrity control is a prerequisite for implementing confidentiality control.

In outline the basis of trust laid down by the system policy is that individuals have two authorisation attributes, a sensitivity label and their identity, and all objects manipulated by individuals similarly have two attributes, a sensitivity label reflecting their contents and a history of the identities of individuals who have modified them. Confidentiality says that in order to observe the contents of something on the system the observing individuals sensitivity label must dominate the observed objects sensitivity label. Integrity says that in order to modify something the modifying individual must not be involved in the history of the object.

Obviously there are supporting requirements to these, for example that labels are updated to reflect contents and history of modifiers etc, but these are the main pillars of the systems policies basis of trust.

In practice there is a third area of security which in reality is totally at the discretion of the individuals as to how well it serves their purpose. This is therefore usually referred to as discretionary security which doesn't mean that it is not enforced by the system.

2.1.5 DISCRETIONARY SECURITY

Discretionary security is in reality an extension to the concept of the original "do it yourself if you want it done right", isolation policy. If I generate information on the system marked in a manner such that only I can observe or modify it then, provided I have assurance that this works, I do not need to concern myself with the adequacy of the systems confidentiality or integrity controls.

This is simply the notion of access control list for objects, listing the identities of individuals who can observe or modify them. If the system creates all objects such that the creators identity is the only one present in the initial list for the object then no one else can access the object. Those who wish or have to risk the threats of sharing and cooperating with others can extend the access control list of objects "safe in the knowledge" that the confidentiality and integrity constraints of the system will limit their risk exposure.

In reality, unless the individual built the system from its RadioSpares parts, toggled in every binary bit of a monitor and bootstrapped up to the final system of compilers and application task software, the individual is dependent on the classification and separation of duty aspects of all those involved in producing and running the system he is using, even a standalone PC.

However, discretionary security can be used within a system policy of confidentiality and separation of duty to enforce useful mandatory aspects of an application policy. For example, the cheques in the Bank example have to be countersigned by the Bank manager, not just someone who is not a counter clerk, such as the catering staff! This is an additional application specific constraint above the system policy of separation of duty.

2.1.6 THE SMITE APPROACH TO TOTAL SECURITY

The SMITE notion of security is for a general purpose computer system, including user programming, which embodies a system policy of confidentialty and separation of duty. Discretionary security is considered an additional application specific aspect which does not violate the system policy. The model presented in section three includes a single access control list element to show how this feature integrates with the system policy.

2.1.7 ASSURANCE OF SECURITY

It is rather a long time since we have heard any views from the Clapham Omnibus as opposed to proponents of this or that model. So....

assurance n. **1.** a statement, assertion, etc, intended to inspire confidence. **2.** a promise or pledge to support. **3.** freedom from doubt; certainty. **4.** forwardness, impudence. **5.** (Chiefly Brit.) insurance providing for certainties such as death as contrasted with fire or theft.
insurance n. **1. a.** the act, system, or business of providing financial protection for property, life, health, etc., against specified contingencies, such as death, loss, damage, and involving payment of regular premiums in return for a policy guaranteeing such protection. **2.** a means of protecting or safeguarding against risk or injury.

Taking rather liberal interpretations of these terms in the context of computer security we would characterise the current approaches to assurance as being heavily weighted to the provision of certainty by means of insurance, that is a guarantee. For many the Basic Security Theorem of Bell-LaPadula, being an inductive proof, provides an absolute guarantee and any formal notion of security which falls short of this is unacceptable. Our expression of a notion of security emphasises the ultimate reliance on fallable human assumptions of clearances, and shared objectives and provides a method, separation of duty, which increases the probablity of success of those objectives. In this it falls short of an insurance guarantee. However, by expressing the notions clearly and unambiguosly by means of a formal model we can provide assurance in terms of providing a statement or pledge intended to inspire confidence within clearly defined boundaries and dependencies.

The degree to which the differences of these approaches are perceived as low or high assurance must always remain a subjective decision of the reader. Our policy approach may appear as an insurance policy hedged around with small print escape clauses whereas Bell-LaPadula is a bold assertion of guarantee. To continue the analogy however we can argue that in reality the bold policy has been renowned for "not paying up claims" and has led its holders into precipitous action beyond that which could legitimately be expected. In constrast our policy leads to an honest appraisal and acceptance of risks.

The main area of precipitous action is that Bell-LaPadula is really only a cast-iron policy in applications where subjects and objects and their levels are static, with information flowing amongst them according to the simple inductive rules. In any system where these preconditions are violated, which includes all practical systems, the system is in reality dependent on the "correct" behaviour of cooperating "trusted processes". The statement of this correctness and the nature of the dependencies is not addressed at all within the Bell-LaPadula model. It is this major deficiency which our modelling approach attempts to address. The apparently small part played by the inductive nature of the model in our approach is thus not a fault of our notion of security but results from exaggerated emphasis of its role in Bell-LaPadula.

3. SECURITY POLICY MODEL

The SMITE Security Policy modelling approach is a state transition based model which can express notions of security such as government confidentiality, integrity issues, and commercial policies with equal facility and in a coherent manner.

The approach is suitable for use in high assurance systems where proof of consistency with security axioms is required and can be refined to implementation in a straightforward manner for the SMITE architecture.

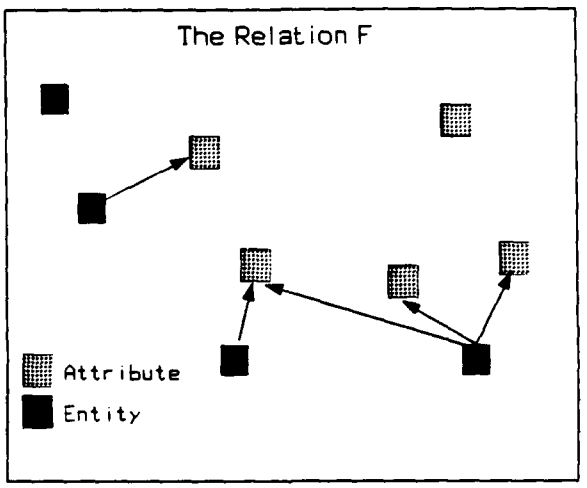
3.1 Basic Modelling Approach

The model is based on a simple state transition machine. It thus consists of two major parts, a definition of the state elements and a definition of the state transition rules.

The model elements are relations on abstract sets of entities and attributes. These are represented by the following Z types.

$$[E, A]$$

The machine can be pictured as a set of entities and attributes with a subset of attributes associated with a subset of entities. There will be some attributes not associated with any entities and some entities with no attributes. The association is a relation in that many entities may be associated with a single attribute and a single entity may be associated with many attributes.



This is the relation $F : E \leftrightarrow A$.

Intuitively these elements are interpreted as representing a computer system as follows. Entities are considered to represent the active and passive components of the system which are visible to each other and can be detectably changed by other entities, for example one may think of active entities as processes and passive entities as directories etc. Attributes are considered to represent immutable components of the system which are only visible, or can be "accessed" by, some subset of entities. Thus if one thinks of attributes as write once files a file may only be accessed via the directory in which it resides. Many directories may have links to a single file etc. The relation F captures the notion of which entities have access to which attributes.

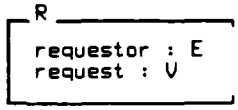
A simple bare machine could be represented as a number of process shells and some memory segments. These would be active and passive entities because they may change. The attributes of the system would be the integer values stored in the memory and process registers. Integers do not change, they are merely replaced by other integers using store operations.

Only entities and attributes in the relation F are considered to "exist" on the system. Adding or removing elements to the domain and range of F is regarded as creating or destroying processes, directories and files etc.

Intuitively one may think of entities containing the addresses of attributes and thus the notion that the only thing that changes in the model is the relation F, and the concept that entities "change" can be linked. The components of a system that exist are addressed by some other component of the system hence the state of the machine at any time can be captured by the relation F. Formally of course attributes and entities are simply inscrutable members of the given sets and do not have a notion of structure or change.

These notions are represented formally by a schema describing the machine state.

The requests that can be made of the machine to change state are thus defined in terms of the state elements and are considered to originate from an entity.

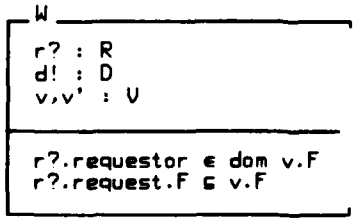


Intuitively this can be read as an entity, "requestor", requesting that the machine state be modified with the state components given by "request". Adopting the notion of a requestor represents the decision that we are modelling purposeful, directed state transitions and not spontaneous processes of nature. The "request" component represents a subset of the machine state accessible to the "requestor" which are essentially the parameters of the request. A request result in three basic state changes. The request may "create" new entities or attributes in F, it may add or remove mappings between existing entities and attributes, or it may "destroy" existing entities or attributes by removing them from F.

The set of decisions that the state machine can make in reponse to a request are defined as

$$D \triangleq \{ \text{"yes"}, \text{"no"} \}$$

The set of valid state transitions that a system can make is represented by a state transition relation defined as



The choice of decision made by the machine in reponse to a particular request in a particular state are considered as subject to state transition rules represented by

```
ρ
r? : R
v,v' : V
```

A system design will define a set of such rules for the system each differing in the constraints expressed in its predicate. For the model to be of use it is important that its behaviour is predictable. To this end the set of rules must define W in such a way that only one rule can cause a state change for any request in any state. Thus permitted transitions are of the following form

```
Transition
W
ω : bag ρ
---------------------------------------------------
∀ rule : ρ | rule.r? = r? ∧ rule.v = v ∧ rule.v' = v' •
  ( ω( rule ) = 1 ∧ d! = "yes" )
∨ ( ω( rule ) = 0 ∧ d! = "no" ∧ rule.v' = rule.v )
```

This states that the system response is "yes" and the resultant state is a request permitted by a single rule, or, if no rule is applicable the response is "no" and no change of state occurs.

The behaviour of a particular run of a system defined by W and an initial state, z0, can be recorded in terms of its sequence of inputs, outputs and states. Such a record is called an appearance of the system.

```
Appearance
permitted : ℙ Transition
x  : seq R
y  : seq D
z  : seq V
z0 : V
---------------------------------------------------
#x = #y = #z

∀ n:ℕ | 2 ≤ n ≤ #x •
   ∃ t : permitted •
      t.r? = x(n) ∧ t.d! = y(n) ∧ t.v = z(n-1) ∧ t.v' = z(n)

∃ t : permitted •
      t.r? = x(1) ∧ t.d! = y(1) ∧ t.v = z0 ∧ t.v' = z(1)
```

We use this modelling approach to describe the required security behaviour of computer systems.

3.2 The Elements

To use the model for our approach to security requires that we refine the components of the system state relation F.

First we require a set of classification/clearance attributes which define the mandatory basis of trust: labels for entities.

$$CLASS : \mathbb{P}\ A$$

Second we require an attribute which designates whether a requestor entity is a process initiated by a user and which may be assumed to contain Trojan Horses or whether it is a trusted process, created by the system, running trusted software which faithfully represents the intentions of the individual; the trusted path.

$$TRUSTED : \mathbb{P}\ A$$

Third we require a set of unique identifiers for individuals for use in separation of duty control. This element is also used for discretionary security aspects.

$$ID : \mathbb{P}\ A$$

Finally, all other attributes are assumed to be information attributes as required to carry out application tasks.

These sets are used by the following relation and function subsets of F.

Class is a partial function giving the classification/clearance of an entity.

Id is a partial function giving the identity of the individual for which requestor entities are acting and other entities have been created.

Trusted is a partial function defining whether a requestor is the individual's faithful proxy or an active entity of unknown predilictions.

Mods is a relation giving the identity of entities which have requested changes to an entity.

Acl is a relation giving the identities of entities which are permitted to observe or change an entity and represents discretionary security aspects.

The system state of the basic model is thus unchanged, but we have provided a nomenclature for examining components of the state.

V

$$\begin{array}{l} F : E \leftrightarrow A \\ Class : E \nrightarrow CLASS \\ Id : E \nrightarrow ID \\ Trusted : E \nrightarrow TRUSTED \\ Mods : E \leftrightarrow ID \\ Acl : E \leftrightarrow ID \end{array}$$

$$F \supset Class \cup Trusted \cup Mods \cup Id \cup Acl$$

The set of classification labels are formed into a lattice in the usual manner.

```
⊤,⊥ : CLASS
_ ≥ _ : partial_order( CLASS )
(_ lub _) : ( CLASS × CLASS ) → CLASS
LUB_ : ℙ CLASS → CLASS
```

```
∀ l1,l2 : CLASS; ls : ℙ CLASS •
  ⊤ ≥ l1
  l1 ≥ ⊥
  l1 lub l2 ≥ l1
  l1 lub l2 ≥ l2
  ∀ l : CLASS | l ≥ l1 ∧ l ≥ l2 •
    l ≥ l1 lub l2
  ∀ l : ls •
    LUB ls ≥ l
```

The sets of attributes are not required to be disjoint. This is because it may be useful to consider that different active entities consider an attribute to be different things. However, it may be necessary to restrict this to make analysis tractable.

3.3 The Axioms

The model is a state transition model and the axioms expressed fall into two interdependent types, state axioms and transition axioms. The state axioms are constraints on the domain of the state elements such that the transition axioms are complete. In all cases the first transition axiom is that the state axiom is upheld in order that the inductive sequence of states is complete. Because the state axioms are upheld inductively by the first state transition axiom the state axioms are effectively constraints on the initial state only.

The model axioms are presented in two stages. For the first stage, each aspect of security, Confidentiality, Separation of Duty, and Discretionary Access, are considered separately. In the second stage the separate axioms are condensed into the "real" security axioms of the model. The axioms for the three areas do not standalone and thus close examination of these in isolation may unsettle ones intuitive notion of security. These standalone versions have been presented to aid initial comprehension and may lose some rigour because of this, thus all close examination should be constrained to the final set of axioms.

3.3.1 CONFIDENTIALITY

Confidentiality is concerned solely with the Class and Trusted subsets of F.

The Secure State Axiom

A general notion of the execution of the model is that only entities and attributes in the relation F "exist" on the system. For this reason Class is a partial function because not until an entity has been "created" with some attribute "contents" can we decide its appropriate classification. For our notion of security control we require that all entities which "exist" are classified so that the transition axioms, which decide appropriate classifications for entities, are complete. This requirement is expressed as a state axiom.

The Secure State Axiom

$$\forall v : V \bullet \mathrm{dom}\ v.Class = \mathrm{dom}\ v.F$$

Trusted is a partial function, not only because of the execution notion above but more fundamentally because not all entities which exist are active entities faithfully representing the behaviour of individuals in the requests that they issue. We have no state invariant or initial state requirement on trust but obviously, as with all initial state aspects, it must be appropriate for the system it generates.

The Secure Transition Axioms

The first transition axiom is trivially that the Secure State Axiom is upheld, that is, that all entities in the new state are classified in the new state.

(i) Uphold the SSA

$$\forall t : Transition \bullet \mathrm{dom}\ t.v'.Class = \mathrm{dom}\ t.v'.F$$

We view a classification as a ceiling on the authority to observe the contents of classified entities which we are modelling as attributes.

In terms simply of our entity and attribute relation F, which subsumes all other aspects of the system, we can define the notion of observe as follows.

A requestor nominates a subset of the machine state in a request for a state transition which results in all of the entities in the domain of the subset being "observed" and possibly some of them being changed. An entity has changed if it has attributes it did not posses before the state change.

Thus our axiom for observe access to entities is as follows.

(ii) Requestor dominates all Observed Entities

$$\forall\, t : Transition;\ e : E \mid e \in dom\ t.r?.request.F \bullet t.v.Class(t.r?.requestor) \geq t.v.Class(e)$$

Untrustworthy entities running on behalf of individuals may include Trojan Horse software which attempts to signal secrets to lower levels by changing the system state in a manner which can be detected by lower level entities. To prevent these contingencies we require a third axiom which uses the trusted notion to restrict all such changes to the trustworthy entities running on behalf of the individual. Changes which can be detected by an entity are the change of attributes of entities it can observe and the creation and deletion of entities in general.

(iii) Untrusted Requestor Does Not Signal Downwards

$$\forall\, t : Transition \mid t.r?.requestor \notin dom\ t.v.Trusted \bullet Higher \lhd t.v'.F = Higher \lhd t.v.F \wedge dom\ t.v'.F = dom\ t.v.F$$

where

$$Higher \triangleq \{e : E \mid t.v.Class(e) \geq t.v.Class(t.r?.requestor)\}$$

This axiom can be circumvented by an untrusted entity modifying the trust relation of the system to mark itself or some other entity as trusted. Therefore we must include the notion that only trusted entities can engage in transitions which modify the trust relation.

(iv) Untrusted Requestors do not Change Trusted

$$\forall\, t : Transition \mid t.r?.requestor \notin dom\ t.v.Trusted \bullet t.v.Trusted = t.v'.Trusted$$

These axioms do not prevent an individual from entering secrets and downgrading them to unclassified, nor do they prevent an individual raising his or someone elses clearance, if such a transition rules exists in the system. This may seem a little radical and insecure. Note however that an individual by the very notion of his clearance is trusted not to declassify classified material so this is maybe not so radical for most downgrade opportunities. However, most real world policies would not let a single individual make a radical change, say more than one level, or a change in material above a certain level without confirmation from another individual. This is concerned with the protection of the integrity of classification labels against fraud by a subverted individual. Thus this is properly the concern of separation of duty and not confidentiality axioms. Similarly an individuals clearance is not a matter of his own choosing but is subject to separation of duty.

3.3.2 SEPARATION OF DUTY

Separation of Duty is concerned with the Id and Mods subsets of F.

The Secure State Axiom

Entities are created on the system at the behest of individuals in order to achieve their alloted task in cooperation with other individuals. Thus all entities can be considered to be "owned" or "instigated" by an individual. The partial function Id is the model representation for determining this identity. The function is partial because not all entities "exist" however we require that all entities that exist are attributed an individuals identity.

Mods is a relation giving the identities of all individuals whose entities have been involved in changes to the entity. Creation of an entity is counted amongst our definition of a changed entity, thus all existing entities have both an Id and a Mods giving us our secure state axiom as follows. Note that the initial Mods of a created entity does not necessarily include the same identity as the Id.

The Secure State Axiom

$$\forall v : V \bullet$$
$$\text{dom } v.Id = \text{dom } v.F$$
$$\text{dom } v.Mods = \text{dom } v.F$$

The Secure Transition Axioms

As usual we must uphold the SSA.

(v) Uphold the SSA

$$\forall t : Transition \bullet$$
$$\text{dom } t.v'.Id = \text{dom } t.v'.F$$
$$\text{dom } t.v'.Mods = \text{dom } t.v'.F$$

Separation of Duty says that changes to an entity can only be undertaken by an individual who has not been previously "involved" with changes to the entity. We can observe that there are many different interpretations of the word "involved" which lead to separation of duty constraints of varying "strength".

For example, if we simply record in the Mods of an entity the identity of all requestors, obtained by Id, who have changed the entity and require that the identity of the new requestor, also by Id, is not in that list we have an apparently strong definition of separation of duty.

However, if the requestor has been instigated under a new Id by a previous modifier of an entity we would in the real world regard him as a "front man" instigated soley to avoid our check. It may be thought that we could require an axiom that all transitions maintain the correctness of Id, that is a process which allocates another process cannot pretend that the new process is someone elses. This is precisiely the transition made by the "login" process of a general purpose computer system, which is why we said that Id and Mods of a new entity are not always the same. Therefore a stronger definition of separation of duty is that the list of those involved with the requestor entity, its Mods, must not share any members with the list of those involved with the entity to be changed, also by Mods. Thus we have not just a "broker" but an "honest broker". This approach would not let the system log on the programmer who wrote the login program, or any other system programmer who wrote any of the software involved in getting from the bootstrap stage to the running of the login process!

There is an even stronger form of separation of duty. If the attributes which a requestor uses to change an entity are obtained from entities which are involved with that entity some form of influence may be exerted on the "honest broker" requestor. That is, if those involved with an entity can restrict the choice of changes the honest broker can make separation of duty may still be subverted. Thus for the greatest assurance we require an honest broker who is subject to no preconditions by either side. This form of separation of duty would not allow the login process to log on someone who had changed the datafile read by login in order to assign initial clearances etc to the new entity.

This, the strongest form of separation of duty requires a history of changes defined as follows.

(vi) Record History of all Changes

```
  ∀ t : Transition •
   t.v'.Mods( Changed ) =
       t.v.Mods( Changed          ) ∪
       t.v.Mods( {t.r?.requestor} ) ∪
       t.v.Mods( Observed         )
where
  Changed ≙ {e : E | t.v'.F({e}) ≠ t.v.F({e})}
  Observed ≙ dom t.r?.request.F
```

Under this definition the history of changes of an entity recursively includes all the identities of those who changed the requestor and not simply the requestor.

Strict Separation of Duty uses this history as follows.

(vii) Strict Separation of Duty

```
  ∀ t : Transition •
   t.v.Mods( {t.r?.requestor} ) ∩ t.v.Mods( Changed  ) = {}
   t.v.Mods( {t.r?.requestor} ) ∩ t.v.Mods( Observed ) = {}
   t.v.Mods( Observed         ) ∩ t.v.Mods( Changed  ) = {}
where
  Changed ≙ {e : E | t.v'.F({e}) ≠ t.v.F({e})}
  Observed ≙ dom t.r?.request.F
```

Unfortunately this definition is too strict!

Consider a secure initial state. The initial process has an Id and a Mods. The only possible behaviour which it can engage in under this definition is to observe himself, a transcendental meditation fanatic. While a life of meditation is fine for Nirvarna back on earth we have to get things done. In order to achieve this we can relax the strict interpretation of separation of duty, or, relax the definition of the labelling requirement, or, both.

Note that these relaxations involve the transition rules being exempt from aspects of the axioms and not individual requestors.

For each transition rule in a system design which is relaxed the designer must be prepared to show, and the procurer agree, that the relaxations are justified in pursuance of the procurers stated requirements, and that no transitions counter to the procurers requirements are accidentally admitted.

Thus the definition of Strict Separation of Duty is a good starting point to negotiate down from, if you have to. The task of enumerating all state transitions and making these justifications may seem so onerous and error prone for a complex system, that the notion of some form of separation of duty adds nothing to assurance of security. Firstly, we can admit this is true but it is true of a Bell-LaPadula system where the formal guidance ends at "if levels change then its a trusted process", with no means of assessing the complex interactions of many trusted processes, each making intuitively "safe" exceptions to the axioms. Secondly, and more constructively, we will show that the use of typing as an approach to structuring the entity-attribute relations, and their associated transitions, does increase assurance of an otherwise complex task.

For confidentiality we appealed to the notions of separation of duty to act as a basis of trust which upholds elements of the system upon which confidentiality rests, integrity of the sensitivity labels. Relaxing the separation of duty constraints for transitions similarly requires a basis of trust to which we can appeal. This is commonly vested in the final authority of a particular individual or group of individuals.

Consider the example of the initial assignment of clearance to individuals from the original confidentiality aspects of the model. This is usually carried out by cleared individuals, who may be lower than the clearance which is to be assigned to the individual under investigation, who have no connection with the individual, and who are the delegated individuals whose job is to assign clearances to people. This last aspect is important. By the particular identity, Jones of the Ministry of Clearances, bolstered by separation of duty and some minimal degree of trust, it is possible for a clearance to be assigned. A lowly cleared, independent teaboy from the Ministry of Agriculture and Fisheries, could not grant an individual a clearance even though he satisfies separation of duty, confidentiality, and trust.

This aspect of security is commonly called discretionary security.

3.3.3 DISCRETIONARY ACCESS

Discretionary security is commonly associated with the notion of a requestors identity and a list of permitted identities associated with each accessed entity, an Access Control List. It is called discretionary because deciding the contents of an Acl is left to the discretion of some individual. Note this discretion is not for any individual but "some" individual. Thus altering an Acl is subject to some other Acl etc. It is precisely because of the difficulty of predicting the possible behaviour of a system based entirely on the discretion of a large number of individuals modifying Acl's that the "loose" control notions of classifications/clearances and separation of duty were introduced. The fact that we end up appealing to such notion in order to close a notion of confidentiality dependent on a notion of separation of duty appears to imply that the notion of security we have defined is based on a circular argument. This is not so only on a "well designed system" where the individuals responible for closing the systems notion of security are not the same as the individuals being subjected to those controls, an ultimate application of separation of duty. On a well designed system the notion is a hierarchy with the top individuals ultimately trusted and responsible for the safe operation of the system. If the system manager is also one of the delegated workers the fact that a model of this is a vacuous circular argument should not come as a surprise because that is exactly what the security of such a system is - vacuous.

For those who find this notion of security a little radical it is possible to add further constraints which reduce the assurance task by reducing the functionality of the system permitted. For example, if the model is constrained such that the Class partition of F doesn't change this model is equivalent to (original) Bell-LaPadula [Bell73a]. We will now formalise the notions of discretionary security and then combine the axioms of all three aspects of security into a single set of axioms.

Discretionary access is concerned with the Id and Acl partitions of F.

The Secure State Axiom

For simplicity we will consider a simple Acl for all entities which when initially created contains only the Id of the created entity. Inclusion of another identity in the Acl by the "owner" allows the new identity equal rights to the "owner", that is it too can modify the Acl. It is in this aspect that the example is simpler than the real aspects of discretionary access control.

Acl is a relation because many entities may be granted access. It does not make sense for an entity to have an "empty" Acl as it would simply linger around the system to no avail. Thus our secure state axiom is that all entities are labelled with the identity of an individual and an access control list.

The Secure State Axiom

```
∀ v : V •
    dom v.Id = dom v.F
    dom v.Acl = dom v.F
```

The Secure Transition Axioms

As before our first transition axiom is that all state transitions uphold the state axiom.

(viii) Uphold the SSA

```
∀ t : Transition •
  dom t.v'.Id = dom t.v'.F
  dom t.v'.Acl = dom t.v'.F
```

In an analogous fashion to confidentiality, access to an entity involves observing its contents or modifying its contents. Our simple example of access control requires that the identity of the requestor is in the Acl of all observed and changed entities.

(ix) Requestor is a member of observed entities Acl

```
∀ t : Transition; e : E
|
 e ∈ dom t.r?.request.F
•
 t.v.Id(t.r?.requestor ) ∈ t.v.Acl( {e} )
```

(x) Requestor is a member of modified entities Acl

```
∀ t : Transition •
  ∀ e : Changed  |e ∈ dom t.v.F •
    t.v.Id(t.r?.requestor ) ∈ t.v.Acl( {e} )
where
  Changed ≙ {e : E | t.v'.F( {e} ) ≠ t.v.F( {e} ) }
```

Note that this example provides an all or nothing control from the point of view of the owner of an entity. Once access is provided to other individuals they may freely propagate access to others because modifying the Acl of an entity is not distinguished from modification in general. Combined with the confidentiality axioms the propagation is limited by classification/clearance and Trust aspects. Further constraint is properly the subject of separation of duty control as we are essentially concerned with the integrity of the Acl.

3.3.4 THE COMBINED NOTION OF SECURITY

The Secure State Axiom

This is a straightforward collecting together of the individual aspects.

The Secure State Axiom

```
∀ v : V •
    dom v.Class = dom v.F
    dom v.Mods = dom v.F
    dom v.Id = dom v.F
    dom v.Acl = dom v.F
```

The Secure Transition Axioms

The first of these is a straightforward collection of the axioms required to uphold the secure state axiom.

(i) Uphold the SSA

```
∀ t : Transition •
  dom t.v'.Class = dom t.v'.F
  dom t.v'.Mods = dom t.v'.F
  dom t.v'.Id = dom t.v'.F
  dom t.v'.Acl = dom t.v'.F
```

We next collect together all those aspects of security expressing constraints on the relationship between the requestor and the observed entities.

(ii) Observed

```
∀ t : Transition; e : E
|
 e ∈ dom t.r?.request.F
•
 t.v.Class(t.r?.requestor ) ≥ t.v.Class(e)
 t.v.Mods{{t.r?.requestor}} ∩ t.v.Mods( {e} ) = {}
 t.v.Id(t.r?.requestor ) ∈ t.v.Acl( {e} )
```

This states that the requestor must dominate all observed entities in Class for confidentiality, have had no previous involvement with them in Mods for separation of duty, and be permitted access in Acl for discretionary access.

Similarly we can collect all those aspects of security governing the requestors authority to change objects.

(iii) Changed

```
∀ t : Transition •
  ∀ e : Changed •
   ( e ∈ dom t.v.F
    ⇒
     t.v.Id(t.r?.requestor ) ∈ t.v.Acl( {e} )
     t.v.Mods({t.r?.requestor}) ∩ t.v.Mods( {e} ) = {}
   )
   t.v'.Mods({e}) =
                t.v.Mods({e}) ∪
                t.v.Mods({t.r?.requestor}) ∪
                t.v.Mods( Observed )
where
  Observed ≙ dom t.r?.request.F
  Changed ≙ {e : E | t.v'.F( {e} ) ≠ t.v.F( {e} ) }
```

This states that for all changed entities if the entity is modified as opposed to created the requestor must have had no involvement with it in Mods for *separation of duty and must be permitted in Acl for discretionary access.* A post condition is that for modified or created objects the requestors and observed entities involvement must be recorded in Mods.

These are essentially the main axioms of security. The remaining axioms are secondary axioms supporting the intent of the main axioms by preventing behaviour which subverts the intention of the policy.

The first of these is that creation/destruction of entities and modifications of lower classified entities can be used by untrusted software executing on behalf of an individual to covertly signal information counter to the policy. These transitions are therefore constrained to Trojan Horse free software and hence to the individual who will not signal unless subverted, in which case an individual has better ways of violating policy than covert channels. This is the trusted path concept.

(iv) Untrusted Requestor Does Not Signal Downwards

```
  ∀ t : Transition •
    t.r?.requestor ∉ dom t.v.Trusted
   ⇒
    Higher ◁ t.v'.F = Higher ◁ t.v.F
    dom t.v'.F = dom t.v.F
where
  Higher ≙ {e : E | t.v.Class(e) ≥ t.v.Class(t.r?.requestor )}
```

As this is dependent on the notion of trusted path it is imperative that this designation is not bestowed by untrusted entities upon themselves.

(v) Only Trusted Requestors Change Trusted

```
∀ t : Transition | t.v'.Trusted ≠ t.v.Trusted •
      t.r?.requestor ∈ dom t.v.Trusted
```

In reality this transition would be subject to additional constaints of separation of duty and discretionary security but the requirement that an individual is accountable for designating entities as Trojan Horse free requires Trusted as the basic prerequisite.

Finally for Strict Separation of Duty the observed and changed entities must be uninvolved with each other in addition to mutually uninvolved with the requestor.

(vi) Strict Separation of Duty

$$\forall\, t : \text{Transition};\ oe, ce : E \mid oe \in \text{dom}\ t.r?.request.F \wedge ce \in \{e : E \mid t.v'.F(\{e\}) \neq t.v.F(\{e\})\} \bullet t.v.Mods(\{oe\}) \cap t.v.Mods(\{ce\}) = \{\}$$

3.3.5 SUMMARY

The notion of trusted process as the basis for allowing policy enforcing yet axiom breaking transitions has long been recognised as too broad a control for Bell-LaPadula based systems. The use of separation of duty on such transitions has hopefully been used for such processes in existing systems but by adding the separation of duty notion formally to our model it makes explicit the design decisions which are made and their effect on the system behaviour. Unlike the simple notions of security these aspects of the system design cannot be proven consistent by a simple inductive approach. The model however cannot be said to be less secure than a Bell-LaPadula based model even if the extra assurance of a formally stated but "unproven" control is rejected.

The model has dispensed with the notions of *-property, Hierarchy, Compatability and Tranquility etc of later examples of Bell-LaPadula modelling and has retained the simple elegance of the basic approach.

3.4 The Rules

If we consider the generic case of the attribute relation F : E ↔ A, the state transitions that can be made can be categorised according to their effects on the domain, range, and mapping of the relation.

To aid the intuitive understanding of the appropriateness of transitions individual transitions should consist of the smallest single modification required for modelling events.

The total number of possible modifications can be generalised to the nine combinations of augmenting, diminishing or leaving unchanged the domain and range of F.

RANGE

DOMAIN	=	+	−	≠
=	Propagate Attributes (gain/lose)	Create Attribute	Destroy Attribute	Change Attribute
+	Create Entity	?	?	
−	Destroy Entity	?	?	

Generic Modifications of F

The combinations named are the minimum transitions for carrying out all modifications of F. The four marked query are valid transitions but can be expressed in terms of two of the named transitions applied sequentially, they are thus not "primitive".

In order to preserve the notion of primitiveness of transition rules, the four create/destroy transitions include the notion that the mappings of uninvolved entity-attribute pairs remains unaltered, as again such a transition can be modelled as the sequential application of a create/destroy and a propagate transition if required.

Again in the interests of simplicity the two propagation transitions are limited to the notion that entities lose or gain references to attributes in the possesion of other entities without affecting the relationship of those other entities. The notion of attributes moving from entity to entity is again not primitive as it can be modelled as gain/lose sequential transitions.

Because of the model concept of functional subsets of F which are subject to state axioms, such as Class, it is necessary to have a simple transition which changes an attribute, ie a combined loss and gain of an attribute.

Thus all possible state transitions can be considered as combinations of these seven basic transitions.

3.4.1 THE SEVEN BASIC TRANSITIONS

The Four Create/Destroy Transitions

Creation of entities is defined as a change of state where one or more new entities mapping to one or more existing attributes are added to the system state relation.

CreateEntity

$$r? : R$$
$$d! : D$$
$$v, v' : V$$

$$v'.F \supseteq v.F$$
$$dom(v'.F - v.F) \cap dom\ v.F = \{\}$$
$$rng\ v'.F = rng\ v.F$$

Destruction of entities is defined as a change of state where one or more entities, possibly with some of their attributes are removed from the state relation, with the mappings of the remaining entities and attributes unchanged.

DestroyEntity

$$r? : R$$
$$d! : D$$
$$v, v' : V$$

$$v'.F \subset v.F$$
$$dom(v.F - v'.F) \cap dom\ v'.F = \{\}$$

Creation of attributes is defined as a change of state where one or more new attributes, mapping to one or more existing entities, are added to the system state relation.

CreateAttribute

$$r? : R$$
$$d! : D$$
$$v, v' : V$$

$$v'.F \supseteq v.F$$
$$dom(v'.F - v.F) \subseteq dom\ v.F$$
$$rng(v'.F - v.F) \cap rng\ v.F \subset rng(v'.F - v.F)$$

Destruction of attributes is defined as a change of state where one or more attributes are removed from the state relation but all entities remain and their mappings to the remaining attributes are unchanged.

DestroyAttribute

$$r? : R$$
$$d! : D$$
$$v : V$$

$$dom\ v.F = dom\ v'.F$$
$$rng(v.F - v'.F) \cap rng\ v'.F \subset rng(v.F - v'.F)$$

The Two Propagation Transitions

The notion of entities gaining mappings to attributes is defined as a change of state where new mappings do not involve the gain of new attributes or entities.

GainAttribute

```
r?   : R
d!   : D
v,v' : V
---------------------
v'F ⊃ v.F
dom(v'.F - v.F) ⊆ dom v.F
rng(v'.F - v.F) ⊆ rng v.F
```

The notion of entities losing mappings to attributes is defined as a change of state where mappings that are lost do not involve the loss of attributes or entities.

LoseAttribute

```
r?   : R
d!   : D
v,v' : V
---------------------
dom v.F = dom v'.F
rng(v.F - v'.F) ⊆ rng v'.F
```

The Functional Change Attribute Transition

The concept of attributes changing is defined as a state change where some entities have lost and some have gained attributes.

ChangeAttribute

```
r?   : R
d!   : D
v,v' : V
---------------------
dom v.F = dom v'.F
rng v.F = rng v'.F
{} ⊂ (v.F - v'.F) ⊆ v.F
{} ⊂ (v'.F - v.F) ⊆ v'.F
```

Analysis shows that these seven transitions have mutually exclusive preconditions and can therefore be made the subject of seven basic rules which fulfill the model requirements for Transitions.

3.4.2 UPHOLDING THE SECURITY AXIOMS

In reality a system designer could propose a set of state transition rules of any complexity desired, provided the effort was expended to show that it upholds the axioms of the model. The purpose of considering all transitions to be combinations of a limited number of simple transitions is to aid in the synthesis of the system design in a manner which increases the likelihood of producing a consistent set of rules which can be shown to uphold the axioms.

Ultimately this approach does not reduce the burden of proof significantly but it does increase the confidence that the proof effort will succeed and it does structure the design and proof effort in a logical manner. The advantages of these two points should not be underestimated in a system design of any significant size or complexity.

One obvious logical structuring approach to aid the intuitive understanding of the system is to consider the security requirements for each of the primitive transitions applied to the overt information subset of F.

An Intuitive Structuring of The Transition Rules

The first point to consider is that the primary purpose of creating entities and attributes comprising a system is to achieve some application task. The Class, Id, Trusted, Acl, and Mods subsets are thus all additional paraphenalia consequent upon this primary goal of manipulating information by a number of delegated individuals. In this task it is entities which model the mutable, shared aspects of the system which these individuals manipulate. The individuals are modelled by entities distinguished by the Trusted function.

Thus creating or *destroying* an entity are the primary events for which consequential changes in the Class, Id, Trusted, etc relations are required. The creation/destruction of entities falls into two classes, that concerned with active entities, the individuals and proxies which they instigate, and passive entities, the system structures which the active entities manipulate.

Having established a structure of active entities sharing access to passive entities the task of manipulating information can begin.

This is represented by the creation of attributes by active entities, which may be the distinguished entities representing individuals or proxy processes instigated in turn by them, their propagation around the passive entities and finally their destruction when no longer of use. The bulk activity of this processing, by means of CreateAttribute, DestroyAttribute, LoseAttribute, and GainAttribute, is constrained by the transition rules' observation of the axioms with respect to the existing contents of the controlling attribute relations such as Class, Id, Mods etc.

A small part of the processing activity may involve changing the control attributes themselves, but only as required to enable the main processing tasks to be accomplished without violation of the security policy.

With this simple overall intuitive structure to guide us we will now examine seven basic rules corresponding to the seven transitions and show how they uphold the axioms and investigate what degrees of freedom are left which enable them to still model useful functions of a general purpose computer system. This is not meant to be an exhaustive specification of a computer system but does demonstrate how the approach can be applied in practice.

The Create Entity Rule

The create entity transition rule will exist in many variants representing the combinations of the various degrees of freedom in the control attributes of the *new entity*.

CreateEntity

```
r? : R
v,v' : U
```

```
r?.requestor ∈ dom v.F
r?.request.F ⊆ v.F

v'.F ⊇ v.F
dom(v'.F - v.F) ∩ dom v.F = {}
rng(v'.F - v.F) ⊆ rng r?.request.F

  dom v'.Class = dom v'.F
  dom v'.Id    = dom v'.F
  dom v'.Acl   = dom v'.F

  ∀ e : Observed •
   v.Class(r?.requestor) ≥ v.Class(e)
   v.Mods{{r?.requestor}} ∩ v.Mods( {e} ) = {}
   v.Id(r?.requestor) ∈ v.Acl( {e} )

  ∀ e : Changed •
   v'.Mods( {e} ) =
          v.Mods( {r?.requestor} ) ∪ v.Mods( Observed )

  r?.requestor ∈ dom v.Trusted

  Higher ◁ v'.F = Higher ◁ v.F
  v'.Trusted = v.Trusted
where
  Observed ≙ dom r?.request.F
  Changed ≙ dom(v'.F - v.F)
  Higher ≙ {e : Changed | v'.Class(e) ≥ v.Class(r?.requestor)}
```

Create entity requires a trusted requestor because it may signal through the domain of F. The freedom of labelling Class is therefore unconstrained by the axioms. We may expect variants which include the Bell-LaPadula style no signalling, and modification of Trusted. We may similarly expect variants which allow login effects on Id as well as the normal owner transition. The Acl is similarly used by a number of variants achieving various application effects.

As with all the rules discussed here the above upholds strict separation of duty. The various variants possible will relax this in different manners depending on Class or discretionary security for justification.

The Delete Entity Rule

DestroyEntity

$$
\begin{array}{l}
r? : R \\
v, v' : V \\
\hline
r?.requestor \in dom\ v.F \\
r?.request.F \subseteq v.F \\
\\
v'.F \subset v.F \\
dom(v.F - v'.F) \cap dom\ v'.F = \{\} \\
\\
\quad \forall\ e : Observed \bullet \\
\quad\quad v.Class(r?.requestor) \geq v.Class(e) \\
\quad\quad v.Mods(\{r?.requestor\}) \cap v.Mods(\{e\}) = \{\} \\
\quad\quad v.Id(r?.requestor) \in v.Acl(\{e\}) \\
\quad\quad v.Mods(\{e\}) \cap v.Mods(Changed) = \{\} \\
\\
\quad \forall\ e : Changed \bullet \\
\quad\quad v.Mods(\{r?.requestor\}) \cap v.Mods(\{e\}) = \{\} \\
\quad\quad v.Id(r?.requestor) \in v.Acl(\{e\}) \\
\\
\quad r?.requestor \in dom\ v.Trusted \\
\quad \mathbf{v'.Trusted = v.Trusted} \\
\\
where \\
\quad Observed \triangleq dom\ r?.request.F \\
\quad Changed \triangleq dom\ (v.F - v'.F)
\end{array}
$$

Again this rule requires a trusted requestor because of changes in the domain of F. The variants will therefore be limited soley to the relaxation of the *separation of duty* constraints.

The Create Attribute Rule

CreateAttribute

```
r? : R
v,v' : V
```

```
r?.requestor ∈ dom v.F
r?.request.F ⊆ v.F

v'.F ⊇ v.F
dom(v'.F - v.F) ⊆ dom v.F
rng(v'.F - v.F) ∩ rng v.F = {}

  ∀ e : Observed •
    v.Class(r?.requestor) ≥ v.Class(e)
    v.Mods{{r?.requestor}} ∩ v.Mods{ {e} } = {}
    v.Id(r?.requestor) ∈ v.Acl{ {e} }

  ∀ e : Changed •
    v.Id(r?.requestor) ∈ v.Acl{ {e} }
    v.Mods{{r?.requestor}} ∩ v.Mods{ {e} } = {}
    v'.Mods{{e}} =
       v.Mods{{e}} ∪ v.Mods{{r?.requestor}} ∪ v.Mods{ Observed }

  r?.requestor ∉ dom v.Trusted ⇒
                      Higher ◁ v'.F = Higher ◁ v.F
                      v'.Trusted = v.Trusted

  ∀ oe : Observed; ce : Changed •
    v.Mods{ {oe} } ∩ v.Mods{ {ce} } = {}
where
  Observed ≙ dom r?.request.F
  Changed ≙ dom(v'.F - v.F)
  Higher ≙ {e : Changed | v.Class(e) ≥ v.Class(r?.requestor)}
```

This will appear in two variants, trusted requestors being allowed or not allowed to signal through lower classified entities. Attribute creation cannot affect the partial function mapping of Class, Id and Trusted but can obviously modify Acl and Mods. The post condition for Mods is fully specified so there is no threat with this rule except to Acl. Application oriented variants with preconditions on modifying Acl may therefore be expected.

The Destroy Attribute Rule

```
DestroyAttribute
r? : R
v,v' : V
---
r?.requestor ∈ dom v.F
r?.request.F ⊆ v.F

v'.F ⊂ v.F
dom(v.F - v'.F) ⊆ dom v'.F
rng(v.F - v'.F) ∩ rng v'.F = {}

   dom v'.Class = dom v'.F
   dom v'.Mods  = dom v'.F
   dom v'.Id    = dom v'.F
   dom v'.Acl   = dom v'.F

   ∀ e : Observed •
     v.Class(r?.requestor) ≥ v.Class(e)
     v.Mods({r?.requestor}) ∩ v.Mods( {e} ) = {}
     v.Id(r?.requestor) ∈ v.Acl( {e} )

   ∀ e : Changed •
     v'.Mods({e}) =
               v.Mods({e}) ∪ v.Mods({r?.requestor})

   r?.requestor ∉ dom v.Trusted ⇒
                     Higher ◁ v'.F = Higher ◁ v.F
                     v'.Trusted = v.Trusted
where
   Changed ≙ dom(v.F - v'.F)
   Observed ≙ dom r?.request.F
   Higher ≙ {e : Changed | v.Class(e) ≥ v.Class(r?.requestor)}
```

Destroy Attribute must not violate the secure state axioms but has no freedom to modify Class, Mods, or Id. Variants allowing trusted requestors to signal or not to lower levels, modification of trusted, and application variants for modification of Acl can be expected.

The Gain Attribute Rule

GainAttribute

```
r? : R
v,v' : V
```

```
r?.requestor ∈ dom v.F
r?.request.F ⊆ v.F

v'.F ⊇ v.F
dom(v'.F ~ v.F) ⊆ dom v.F
rng(v'.F ~ v.F) ⊆ rng v.F

  ∀ e : Observed •
    v.Class(r?.requestor) ≥ v.Class(e)
    v.Mods({r?.requestor}) ∩ v.Mods({e}) = {}
    v.Id(r?.requestor) ∈ v.Acl({e})
    v.Mods({e}) ∩ v.Mods(Changed) = {}

  ∀ e : Changed •
    v.Id(r?.requestor) ∈ v.Acl({e})
    v.Mods({r?.requestor}) ∩ v.Mods({e}) = {}
    v'.Mods({e}) = v.Mods({e}) ∪ v.Mods({r?.requestor})

  r?.requestor ∉ dom v.Trusted ⇒
                          Higher ⩤ v'.F = Higher ⩤ v.F
                          v'.Trusted = v.Trusted
where
  Observed ≙ dom r?.request.F
  Changed ≙ dom(v'.F ~ v.F)
  Higher ≙ {e : Changed | v.Class(e) ≥ v.Class(r?.requestor)}
```

Application variants for Acl, trusted requestors being able to signal, and trusted requestors modifying trusted can be expected.

The Lose Attribute Rule

LoseAttribute

```
r? : R
v,v' : V

r?.requestor ∈ dom v.F
r?.request.F ⊆ v.F

v'.F ⊂ v.F
dom(v.F - v'.F) ⊆ dom v'.F
rng(v.F - v'.F) ⊆ rng v'.F

   dom v'.Class = dom v'.F
   dom v'.Mods  = dom v'.F
   dom v'.Id    = dom v'.F
   dom v'.Acl   = dom v'.F

   ∀ e : Changed •
     v.Id(r?.requestor) ∈ v.Acl(|{e}|)
     v.Mods({r?.requestor}) ∩ v.Mods(|{e}|) = {}
     v'.Mods({e}) = v.Mods({e}) ∪ v.Mods({r?.requestor})

   r?.requestor ∉ dom v.Trusted ⇒
                        Higher ⩤ v'.F = Higher ⩤ v.F
                        v'.Trusted = v.Trusted
where
   Changed ≙ dom(v.F - v'.F)
   Higher ≙ {e : Changed | v.Class(e) ≥ v.Class(r?.requestor)}
```

As above but with the constraints of upholding the Secure State Axiom.

The Change Attribute Rule

ChangeAttribute

```
r? : R
v,v' : V
```

```
r?.requestor ∈ dom v.F
r?.request.F ⊆ v.F

dom v.F = dom v'.F
rng v.F ≠ rng v'.F
{} ⊂ (v.F - v'.F) ⊆ v.F
{} ⊂ (v'.F - v.F) ⊆ v'.F

   dom v'.Class = dom v'.F
   dom v'.Mods  = dom v'.F
   dom v'.Id    = dom v'.F
   dom v'.Acl   = dom v'.F

   ∀ e : Observed •
    v.Class(r?.requestor) ≥ v.Class(e)
    v.Mods({r?.requestor}) ∩ v.Mods( {e} ) = {}
    v.Id(r?.requestor) ∈ v.Acl( {e} )
    v.Mods( {e} ) ∩ v.Mods( Changed ) = {}

   ∀ e : Changed •
    v.Id(r?.requestor) ∈ v.Acl( {e} )
    v.Mods({r?.requestor}) ∩ v.Mods( Changed ) = {}
    v'.Mods({e}) = v.Mods({e}) ∪ v.Mods({r?.requestor})

   r?.requestor ∉ dom v.Trusted ⇒
                    Higher ◁ v'.F = Higher ◁ v.F
                    v'.Trusted = v.Trusted
where
   Looser ≙ dom (v.F - v'.F)
   Gainer ≙ dom (v'.F - v.F)
   Changed ≙ Looser ∪ Gainer
   Observed ≙ dom r?.request.F
   Higher ≙ {e : Changed | v.Class(e) ≥ v.Class(r?.requestor)}
```

The variants of this rule basically cover the entire span of Bell-LaPadula trusted process functionality and will exist in the greatest number of variants. This family of rules can change anything in the system subject to the separation of duty, discretionary and optional clearance constraints available when contructing the system design. It is the presence of this rule which enables the full functionality of computer systems to be expressed by the model. The above is the rule with the least preconditions but strict separation of duty. As soon as this is relaxed, unless it is replaced with extremely tight discretionary or another less strong separation of duty constraint appropriate to the application, an insecure system will result. While expressing individually very tightly constrained variants of this rule is easy, the combined effect of a number of variants becomes less predictable.

3.4.3 BASIC SECURITY THEOREM AND PROOF

Given:

$$
\begin{array}{l}
\omega \triangleq \{ \\
\quad \text{CreateEntity, CreateAttribute,} \\
\quad \text{DestroyEntity, DestroyAttribute,} \\
\quad \text{LoseAttribute, GainAttribute,} \\
\quad \text{ChangeAttribute} \\
\quad \} \\
\\
W = \text{Transition}
\end{array}
$$

z0

$$
\begin{array}{ll}
F & : E \nrightarrow A \\
Class & : E \nrightarrow CLASS \\
Id & : E \nrightarrow ID \\
Trusted & : E \nrightarrow TRUSTED \\
Mods & : E \nrightarrow ID \\
Acl & : E \nrightarrow ID
\end{array}
$$

$$
\begin{array}{l}
F \supset Class \cup Trusted \cup Mods \cup Id \cup Acl \\
\\
E = \{\text{"INIT"}\} \\
Class = \{\text{"INIT"} \mapsto T\} \\
Id = \{\text{"INIT"} \mapsto \text{"Init"}\} \\
Trusted = \{\text{"INIT"} \mapsto \text{"TRUSTED"}\} \\
Mods = \{\text{"INIT"} \mapsto \text{"Init"}\} \\
Acl = \{\text{"INIT"} \mapsto \text{"Init"}\} \\
Information = \{\}
\end{array}
$$

$$SYSTEM \triangleq Appearance \times W \times z0$$

THEOREM.
This system is secure in terms of SSA, STA
PROOF
Omitted.

4. THE SMITE ARCHITECTURE

This section examines the fundamental mechanisms required to implement the Security Policy Model. It then introduces the basic architecture of the SMITE multi-processor by first describing the capability addressing nature of the architecture, followed by the basic protection mechanisms used to provide the discrete components of the policy model abstraction, entities, attributes and the state relation. Finally a section describing the precise mapping of the implementation mechanisms to the policy model is described.

4.1 The Basic Requirements of the Model

The basic elements of the policy model abstraction are Entities, Attributes, their Relationships, and the state transition Rules.

In the formal model entities and attributes are inscrutable elements of the sets E and A. There is no formal sense in which an entity or attribute is or contains data or anything else. The only notion of change in the model is the state relation F.

We are interpreting the F relation as representing the notion of access to attributes by an entity. Our interpretation of the model has no concern for what a notion of "access" to an attribute means except that the semantics of access never changes in the face of spatial or temporal multiple access. In real terms this may be thought of as an attribute being a read-only and/or executable thing. The important notion of access is that of an entity being able to "name", "address" or "reference" an attribute and it is simply this which we are capturing in the F relation. Entities cannot name or conceive of accessing attributes which are not in the range of their component of F. The model assumes that all entites can "name", "address", "reference" all other entities which exist in F.

The model representation of a request is that the "requesting entity" presents a subset of the entities and attributes "which it can name", "to which it has access". It is the transition rule which observes, changes and modifies the set of offered entities and/or the requesting entity with attributes obtained from only those named entities according to the constraints of the rule. Thus the state relation is visible only to the transition rules.

In the same way that we are regarding attributes as simply immutable things which can be named, whereas in reality it is a read-only memory segment or disc file containing textual data, we can regard the entity as more than simply an immutable named index value into the centrally stored state relation. The entities can be interpreted as "containing" the names, or references, to the attributes in their component of F. In other words F is the sum of the contents of existing entities. This requires that an entity is represented by an object which can "hide" its contents from entities but not from transition rules.

Active entities in reality will be processes and attributes will be stored in memory. As we need to constrain a process' access to attributes we require memory mapping constraints in our concrete machine. The active entity, a process, must not have visibility of the mappings or be able to effect them but can invoke transition rules which can adjust mappings. This is simply the requirement for a two state machine. Given this, the supervisor state code represents the state transition rules including the code necessary to manufacture passive entity and attribute objects held in a filestore.

The model is therefore a generic security model which can be implemented with any conventional Trusted Computer Base technology.

The architecture proposed for implementing the model in the SMITE project is a capability addressing processor. Experienced readers may wonder why, having shown that a conventional TCB approach is necessary and sufficient for the model's implementation, we are proposing a capability processor. Capability architectures have a bad reputation as architectures for implementing secure systems despite the intuitively attractive features which appear to make them ideal: fine grain, flexible, hardware enforced protection.

We intend to show that this reputation is undeserved and that just as an insecure machine can be built from the same two state architecture of a conventional TCB, the fact that insecure machines have been built from capability architectures should not reflect on the basic architecture.

Our conjecture is that capability machines have failed because they have been applied to complex tasks with only a simple notion of security to guide the implementation. By complex tasks we mean complexity beyond the practical reach of conventional TCB's. In other words we regard the limiting factor as the models that they have implemented rather than the basic architecture. The models can only define security for relatively simple tasks, a capability architecture is more unwieldy to control than a conventional TCB approach for such tasks and is therefore not the architecture of choice. Given a model which permits of complex tasks we conjecture that the conventional TCB approach will grow in terms of grain and flexibility until it becomes a software implementation of a capability architecture.

In pursuing this argument it is important that we avoid confusion between the SMITE meaning of terms, such as capability protection, and the complex concepts with which a reader with some knowledge of previous capability architectures may possess. In order to achieve this we will therefore describe the very simple notions that we use in SMITE before showing its correspondence to the model.

4.2 The SMITE Basic Architecture

Capability addressing is simply the concept that areas of memory are addressed by a pointer. The difference between a capability and say a conventional base or index addressing mode is that a word representing a capability is fundamentally distinguished from a word representing a scalar number. Thus in a conventional architecture a word representing a number can be used in arithmetic instructions by loading it into "accumulator" registers or it may be used in address calculation of memory store and fetch instructions by loading it into a "base" or "index" register.

In a capability architecture the registers and words in store used to address store are typed differently from normal scalar registers and words and the different roles are enforced by the various instructions of the instruction set. The typing of words may be achieved by additonal "hidden" tag bits or by segregation of the store, this being an implementation detail [Wiseman82]. Capabilities also enforce the bounds of the area of memory addressed, by a size or range field within the capability, and thus represent a stronger addressing mode than the conventional base or index register alone where the memory to be addressed is limited solely by the size of the offset register.

The use of capabilities, of a base-range form, as the only mode of addressing and the fundamental enforcement of the distinction between capability and scalar data is the only notion of protection which SMITE implies by the use of capability protection. A capability cannot be forged or guessed by manipulation of scalar data, a scalar word with the same bit pattern as a capability word are not the same, by virtue of the location of the word or by the hidden tag bit in a segregated or tagged architecture respectively.

There are no instructions to "generate" a capability per se in a capability machine, instead instructions which generate new objects generate new unique capabilities to address them. Capabilities can be freely copied to enable sharing of objects.

Thus far SMITE and previous capability approaches share common concepts. The differences between SMITE and other capability architectures arises from the degree and method of use of the additional mechanisms which are built onto the basic capability pointer and protection idea. A generic development of these mechanisms and the contrasting uses of SMITE and conventional architectures is developed below simply to provide the necessary vocabulary for discussing the SMITE implementation of mechanisms corresponding to constraints of the policy model.

Words used as capabilities are essentially private data types of the machine instruction set used in address calculation. Because of this private nature capabilities and their instructions can be further "typed" by using "spare" bits of the word, ie those bits not required to form a pointer. This and other exotic uses of the bit encodings of capabilities has been used extensively in past capability systems. Thus bits have been used to provide read, write, and execute access control on the store addressed by the capability, bits have been used to control entry and exit from "supervisor" or "privilege" states of code accessed by execute access capabilities, and bits have been used to provide extensive typing to produce high-level object based language processors [Tyner81]. Past efforts to turn the access control nature of read write access capabilities into full blown policy enforcing architectures, with security labels and built in "dominates" rules in the instruction set, have also been attempted.

SMITE uses the "extra" bits to define a number of different types of capabilities. SMITE uses the term "block" and "block type" to denote this on the basis that in practice a capability defines some block of store in terms of a base and offset. Thus the instruction set is partitioned into instructions which work on different capability types where the type defines the format and interpretation of the contents of a block [Currie85, Cooper87].

SMITE further defines a single bit for a capability status, "Locked" or "Unlocked". For each capability type the influence of the lock bit is defined as a modification on the permitted access to the block contents. For all types unlocked denotes unconstrained access to the blocks contents. For some types locked functions as a modification inhibit bit, for other types it may act as a total bar on access to the block contents. In both cases the protection status conferred in the locked state may extend to the entire contents of the block or only to some initial number of words of the block. The instruction set creates capabilities to blocks in the unlocked or open state and provides an instruction for locking a capability. (NB, lock is a capability status bit not a block status). There is an instruction for unlocking a capability, but this only applies to certain types of block in special circumstances.

In such an architecture Entities and Attributes are capability addressed blocks. The Relationship between Entities and Attributes is represented by the capabilities of Attributes stored in Entity blocks. Only a subset of the capability addresses are equated to Entities and Attributes which may thus posses capabilities to other blocks which have no corresponding element in the policy model. These blocks can therefore be regarded as the primitive building blocks from which Entities and Attributes are built. The policy model and its axioms are built on the notion that Entities and Attributes are discrete, independent items related only in the ways explicitly defined by the model. There is thus an obligation to show a proof of separability on the capabilities used to build Entities and Attributes.

The instruction set provides mechanisms to manipulate capabilities to blocks and the contents of blocks in a primitive manner which cannot be cognisant of the policy model versus primitive permitted operations. We therefore require "blocking" or "hiding" mechanisms to protect the policy model mappings from arbitrary instruction set manipulation and yet to allow the application of policy model rule transitions, which can only be sequential applications of individual instructions of the primitive instruction set themselves stored in a capability addressed block.

We will develop the semantics of the required hiding mechanism in a rather tutorial fashion so as not to introduce any false connotations.

Define a block type for which the capability LOCK bit semantics are any access versus no access. If the locked capabilities to these blocks are used for representing Entities no one can alter the relationships or obtain copies of the attribute capabilities contained therein. In addition if these locked capabilities are also used to represent Attributes arbitrary modification cannot violate the separability constraints required of both with respect to primitive blocks.

We require that a sequence of instructions which represents a state transition rule can access the block and manipulate the block contents.

Note it is the state transition rule which must access the block not the instigating active entity, the requestor in model terms. Thus we require that the transition rules are represented in some way which combines not only the constraints, ie the particular sequence of instructions, but protected access to entities, ie only they posses the open capabilities. This representation must exist as a capability addressed block within the system and thus the open capability within it must be hidden from arbitrary access which could steal it and carry out unconstrained transitions on the model relationships.

Thus the transition rule representation must have capabilities whose semantics are that only invocation, or execution, is allowed. These blocks in SMITE are called closures.

A closure block contains two words, a capability to a block containing scalar and capability words, and a capability to a block containing scalars which are interpreted as instructions.

The principle property of the closure type is that closures can only be invoked and that the two pointers are hidden and inaccessible given only a capability to the closure block. Conversely, the only view of the system available to a closure when invoked is its own scalar/capability block and the parameters supplied by the caller.

In the closure regime sensitive values, such as the open capabilities to entities and attributes, required by code, such as the transition rules, are stored in the scalar/capability block. This is used by the transition rule code when invoked and is inaccessible to the caller, before, during and after the call. The code of a closure, being software, is almost infinitely variable in the exercise of checks that can be made upon the parameters of the call in deciding whether to proceed with the access to the sensitive data structure.

The closure scalar/capability block and code block form a tailored protection environment upon each call and thus serve in the same way as protection rings, supervisor states etc of two state machines except that they are not fixed in either form, function or number by the system but exist in a flexible, distributed form. This approach was correctly identified in the Plessey 250 capability machine [England75], where the enter capability was used to structure system software in the same way. Sadly this effort failed primarily for non-technical reasons though as we shall see below there are some other aspects of the SMITE system which the System 250 lacked.

An implementation conformant with the policy model could be mounted with these as the basic protection mechanims, a black-box block and closure containing the open capabilities to the contents of the black-boxes. The system would probably have to be fairly static in terms of the creation and destruction of entities and attributes because the system structuring requirements to instantiate closures containing capabilities to the new objects without security compromise can become onerous with only closure and black-box protection.

The Plessey 250 carried out this instantiation at link time for the system build using a cumbersome link language. In SMITE the interpretation of execution access protection is fully supportive of Landin's Closure notion, which includes run time support.

The concept of the Closure based software regime is simply that the code of a procedure is bound to its environment, defined by its code, constants and non-local variables, to form an independent executable unit which may be applied to parameters but more importantly can be stored and manipulated as for any other data. Thus it may be passed as parameters and returned as results and stored in data structures giving rise to a number of elegant language and software properties that give the concept another name: first class procedures, [Currie82].

This is simply implemented on SMITE with the closure scalar/capability block containing the non-locals and the code block the procedure body. The ability to mix capability and scalar data within a block is an advantage of a tagged capability architecture such as SMITE which the segregation architecture of the System 250 lacked. Absence of this feature adds an extra level of complexity when mapping a procedures non-locals, which may include scalars and capabilities, making a simple implementation of the pure software closure concept difficult.

Even with this high level of run-time support on SMITE the instantiation problem for new black-boxes becomes onerous because of problems which arise from a system containing both trusted and untrusted code. The problem is essentially that trusted and untrusted closures cannot be differentiated thus allowing spoofing during the distribution of the new closures. In order to mitigate this problem the basic black-box mechanism is extended to a notion of typed objects.

Instead of instantiating a set of transition rule representations for access to each instance of black-box we provide a single set which can accept the locked capability to a black-box as a parameter and then "open" the black box capability within the protective domain of the closure environment. This requires the provision of an unlock instruction to produce open copies of locked black-boxes. In order that we don't regress to the problem of how to stop arbitrary instruction sequences opening any black-box the unlock instruction for black-boxes is "keyed" to some unforgeable token passed by closures which are intended to access black-boxes. This is achieved by the following sematics for Keyed Blocks, the SMITE name for such selectively opened black-box capabilities.

A keyed block capability can be locked by any one but can only be unlocked by someone who can quote the first word of the contents of the block, the "key". This word cannot be forged, or guessed, if it is a capability for a block which is known only to the intended closures. For the purposes of the model, this could be a system wide unique key passed by all state transition closures and used to lock all entities and attributes. For integrity policies, the transition rules will naturally partition into groups of rules concerned with partitioned subsets of entities and attributes. This is normally expressed in the model by means of attributes for distinguishing such "types". It is relatively obvious therefore to use the key itself as the representation of such type attributes in implementing the model. This provides the damage limitation notion of least privilege in the implementation of the transition rules.

The instantiation of the transition rules is carried out by a closure which generates a block to get a "key" and then delivers the transition rule closures with this key as a non local. Provided that the code of this initial closure, and the transition rules delivered, never deliver the "key" capability, or, open pointers to blocks containing the "key", as a result, the contents of all blocks accessed by the locked capabilities are immutable and hidden to all but the transition rules.

This is then the sum total of the SMITE architecture requirements for implementing the model with fine grained, flexible, hardware enforced protection: capability addressing, Open/Lock capability status, Closures, Keyed Capabilities.

4.3 Implementing the Model Elements

4.3.1 Introduction

The implementation of each of the policy model elements is discussed below in terms of the essential behaviour requirements implied by the policy model, in other words the constraints of the refinement proof chain.

4.3.2 Attributes

The essential nature of attributes is that they are immutable, tranquil objects. Once created there must be no primitive sense in which they can be modified. There are no constraints other than this on the complexity of attributes in terms of the primitive objects from which they are built.

The tranquility requirement is required so that when transition rules incorporate attributes into entities as relationships there is no variety that can be imparted to subsequent state transitions involving the attribute. For this reason state transitions require an easy and reliable method to identify primitive structures as valid candidates for use as attributes.

Implementing attributes as a read only primitive capabilities is not sufficient to ensure tranquility because a read only capability to the root of a capability structure does not imply that all elements within the structure are read only. Furthermore there is no way to know that the subject has not retained write capabilities to the object or its elements.

Thus we use trusted closures which create attributes by copying a primitive capability structure into a read only copy within a type protected object. The copy is unique, because only the creating closure can possibly have write capabilities to it, and tranquil, because this closure will not use those capabilities to alter the attribute and will deliver only the locked capability to the containing typed object. It is thus labelled as tranquil which can quickly be identified by state transition rules. As for the basic protection, the use of many keys for typing attributes instead of a single tranquility key and additional type information is an optional embellishment and least privilege mechanism.

The untrusted code of active entities can obtain a read only copy of the primitive structure inside an attribute, for use in "algorithmic manipulation" outside of the model, by means of complementry trusted closures which deliver another copy. If the attribute creating closure stores "read only" capablities to the primitive copy in the attribute the second copy by the complementary closure can be optimised to delivery of the read only capability.

These closures may exist as a separate notion from the CreateAttribute family of state transition rules or incorporated into them as a subroutine but this trusted functionality is required to uphold the separability proof obligations on attributes. The "reading" of the attribute contents is an extra model notion which is not considered within any state transitions, but is required for the implementation to uphold the explicit requirement of the model that attributes are immutable.

4.3.3 Passive Entities

In terms of its primitive structure a passive entity is little different from an attribute, a capability structure within a typed object. An entity however is exempt from the tranquility requirement in as much as the state transition rules will define permissable circumstances in which an entity may be modified.

The exact choice of the appropriate capability structure within the object, such that it may fulfill the sematics of the model's state transitions but does not allow other transitions inadvertently, is of critical importance to the successful implementation of a secure system.

The sematics of an entity is implementable purely with a structure of a vector of capabilities to attributes. For state transition/entity "types" which imply a complex sematics this is obtained at the cost of complexity in the code of the transition rules in finding attributes within the entity for the particular transition required. It may therefore provide higher assurance by structuring the attribute capabilities within the entity object. Great care must then be taken however that the state transitions do not provide "visibility" of any variety in the structure over and above the sematics of the type.

4.3.4 Active Entities

An active entity is a closure in the workspace chain of a launched process block. Not all closures in the workspace chain of a launched process block are active entities.

Within a process block is a pointer to the current workspace block. This in turn contains a pointer, in a permanently hidden area, to the current closure. In the open area of a workspace block are the locals and expression stack of the closure. A closure may obtain pointers to its local workspace area and its non-locals and is free to use these primitive pointers in any way it sees fit.

Capabilities to Entities, Attributes and primitive data may be stored in any manner within these two areas and may be manipulated by the closure at will.

For implementing the security policy constraints as opposed to the basic model it is important that the control relations and attributes of active entities such as Trusted, Class etc cannot be modified by the untrusted code of the entity but only by invoked state transitions.

Thus within the process block is a pointer to a table called the "Process Context". This table is a vector of word pairs where the first word of a pair is considered a "name" and the second word the resolved value. If the second word is a capability arbitrarily complex values can be resolved. If the first word is a capability then the name cannot be forged. Instructions are provided which store a new pair resolve a word given a name. Storing a value with a name that all ready exists in the table is an update.

Any closure can store a word with a new name in this table but only closures which possess the existing name can update and/or retrieve words from this table. Thus the control Attributes, which must not be "lost" or substituted by the untrusted code of the active entity, are stored in this area with capability names known only to the transition rules.

A process is launched with an empty context table and an initial closure. This initial closure must be a trusted closure which will effect either a CreateEntity state transition which will label the entity, or arrange to inherit the context of the parent process, before running the intended closure. This intended closure is the entity and can invoke other closures resulting in a new workspace and an inherited context. These new closures are thus not entities but just the code of the entity. If the closure calls a state transition which embodies a CreateRelation transition the new closure will be running as a new entity with a changed process context. When this new entity exits the process context must be restored to that of the existing entity. Thus the process context must be carried in and restored on exit from closure invocations. This is simply acheived by caching the process context table in the chain of workspace blocks of the process.

There are thus two methods of allocating new active entities, as subtasks in an existing process or as parallel tasks in a new process. A subtask or a parallel task is not necessarily a new entity. In all four cases trusted closures are required to ensure the correct labelling in terms of the model transition which is required.

Capabilities within an active subject will include, in addition to primitive blocks, those to closures, attributes and other entities which can be passed as parameters and received as results to and from closures. Closures which are not transition rules are considered primitive and are subject to the separabilty proof obligations.

4.3.5 Transition Rules

Transition rules are closures.

5. FUTURE PLANS

5.1 The Security Policy Approach

The concept of confidentiality, separation of duty, and individual accreditation as three interdependent aspects of security seems to us to be a sound approach. The adoption of the Bell-LaPadula state transition model approach was adopted as a well known framework which held out the promise of achieving the same inductive Basic Security Theorem type approach for the confidentiality subcomponent of our model. This notion does not stand up in our model, not because of any shortcoming in the approach, but because it puts the limitations of the inductive proof in perspective for Bell-LaPadula models. There is therefore some scope for recasting the model in some other formalism which may provide other advantages providing the rigid attitude to what constitutes a "proven and acceptable" policy model can be overcome.

Further work on such other formalisms is required in order to assess the balance of advantage which accrues. Non-interference, deducability, and restrictiveness offer advantages in a distributed system architecture but seem to us to be best used as an analysis technique for establishing useful properties of a design as opposed to constructive techniques. Thus the use of state transition models will always be required at some point in a system design.

5.2 The Policy Model

The model presented in section three is basically the most austere and restrictive form of our approach to security and represents a first attempt at formally expressing it. Further work to find the best method of expressing and demonstrating the approach is required. Only preliminary thought has been expended on the problems of generating realistic rules with relaxed versions of the axioms in a manner which exploits the interactions of the three aspects of security. This has identified that the structuring of state transition rules and the entities and attributes which they manipulate by use of the Abstract Type paradigm may make the assurance task tractable. A useful exercise in this direction would be to recast the RSRE SERCUS specification [Harrold88] in terms of the policy model.

For a practical model the useful technique of exploiting the dual nature of sensitivity labels described in [Woodward87] could be utilised.

REFERENCES

[Andrews81] Andrews,M.P., et al, "Multilevel Computer Security in Network Environments", Plessey Telecommunications, 1981.

[Barnes85] Barnes,D.H., "Secure Communications Processor Research", Procs. MILCOMP 85, London, October 1985.

[Bell73a] Bell,D.E. & LaPadula,L.J., "Secure Computer Systems: Mathematical Foundations", MTR-2547, Vol 1, Mitre Corp.,November 1973.

[Bell73b] Bell,D.E. & LaPadula,L.J., "Secure Computer Systems: A Mathematical Model",MTR-2547, Vol 2, Mitre Corp.,November 1973.

[Bell74] Bell,D.E., "Secure Computer Systems: A Refinement of the Mathematical Model",MTR-2547, Vol 3, Mitre Corp.,April 1974.

[Bell76] Bell,D.E. & LaPadula,L.J., "Secure Computer System: Unified Exposition and Multics Interpretation", MTR-2997,Mitre Corp.,March 1976.

[Biba77] Biba,K.J., "Integrity Considerations for Secure Computer Systems", ESD-TR-76-372, Electronic Systems Division, AFSC, Hanscom AFB, MA, April 1977.

[Clark87] Clark,D.D., & Wilson,D.R., "A Comparison of Commercial and Military Computer Security Policies", Proc. 1987 Symposium on Security and Privacy, April 1987.

[Cooper87] Cooper,D.G. & Diss,M.J., "The Specification of the SMITE Instruction Set in Z", Report Number 72/87/R528/U, Plessey Research Roke Manor Limited, December 1987.

[Currie82] Currie,I.F., "In Praise of Procedures" RSRE Memorandum No. 3499, 1982

[Currie85] Currie,I.F., Foster,J.M. & Edwards,P.W., "PERQFLEX Firmware", RSRE Report 85015, 1985.

[England75] England,D.M., "Capability Concept Mechanisms and Structure in System 250", Rev. Fr. Autom. Inf. Rech. Oper. (France), Vol 9, Sept 1975.

[Foster82a] Foster,J.M., Currie,I.F. & Edwards,P.W., " Flex: A Working Computer With An Architecture Based On Procedure Values", RSRE Memorandum 3500, 1982.

[Foster82b] Foster,J.M. & Currie,I.F. "CURT: The Command Interpreter Language for Flex", RSRE Memorandum 3522, 1982

[Foster83] Foster,J.M., Currie,I.F. & Edwards,P.W., " Kernel and System Procedures in Flex" RSRE Memorandum 3626, 1983.

[Goguen82] Goguen,J.A. & Meseguer,J., "Security Policies and Security Models", Procs. Symp. Security and Privacy, Oakland,Ca, 1982.

[Harrold88] Harrold,C.L., "Formal Specification of a Secure Document Control System for SMITE",RSRE Memorandum No. , 1988.

[Landin64] Landin,P.J., "The Mechanical Evaluation of Expressions", Computer Journal, Vol 6, Num 4,Jan 1964.

[McLean87] McLean,J., "Reasoning About Security Models", Proc. 1987 Symposium on Security and Privacy, April 1987.

[Redell74] Redell,D.D., "Naming and Protection in Extendible Operating Systems", MIT Technical Report MAC-TR-140, Nov 1974.

[Rushby81] Rushby,J.M., "Design and Verification of Secure Systems", ACM Operating System Reviews, Vol15, Num5,Dec 1981.

[Spivey87] Spivey.,J.M., "The Z Notation: A Reference Manual", Draft JMS-87-12a, Programming Research Group, Oxford University, 1987.

[Tyner81] Tyner,P., "iAPX-432 General Purpose Data Processor: Architecture ReferenceManual", Intel Corp., January 1981.

[Wiseman82] Wiseman,S.R., "Two Advanced Computer Architectures: A Study of their Support for Languages and Operating Systems" RSRE Report 82013, July 1982.

[Wiseman86a] Wiseman,S.R., "A Secure Capability Computer", Procs. IEEE Symp. Security and Privacy, Oakland CA, April 1986

[Wiseman86b] Wiseman,S.R., "A Capability Approach to Multi-level Security" Procs. IFIP/Sec, Monaco, December 1986

[Wiseman88a] Wiseman,S.R., "Protection and Security Mechanisms in the SMITE Capability Computer", RSRE Memorandum 4117, 1988.

[Wiseman88b] Wiseman,S.R. & Field-Richards,H.S., "The SMITE Computer Architecture", RSRE Memorandum , 1988.

[Wiseman88c] Wiseman,S.R., Terry,P.F., Wood,A.H., & Harrold,C.L, "The Trusted Path between SMITE and the User" Procs. IEEE Symp. Security and Privacy, Oakland CA, April 1988

[Woodward87] Woodward,J.P.L., "Exploiting the Dual Nature of Sensitivity Labels",Proc. 1987 Symposium on Security and Privacy, April 1987.

DOCUMENT CONTROL SHEET

Overall security classification of sheet ...UNCLASSIFIED.....

(As far as possible this sheet should contain only unclassified information. If it is necessary to enter classified information, the box concerned must be marked to indicate the classification eg (R) (C) or (S))

1. DRIC Reference (if known)	2. Originator's Reference	3. Agency Reference	4. Report Security Classification
	Memorandum 4188		Unclassified

5. Originator's Code (if known)	6. Originator (Corporate Author) Name and Location
7784000	Royal Signals and Radar Establishment St Andrews Road, Malvern, Worcestershire WR14 3PS

5a. Sponsoring Agency's Code (if known)	6a. Sponsoring Agency (Contract Authority) Name and Location

7. Title

ON THE DESIGN AND IMPLEMENTATION OF A SECURE COMPUTER SYSTEM

7a. Title in Foreign Language (in the case of translations)

7b. Presented at (for conference papers) Title, place and date of conference

8. Author 1 Surname, initials	9(a) Author 2	9(b) Authors 3,4...	10. Date	pp. ref.
Terry P F	Wiseman S R		1988.6	54

11. Contract Number	12. Period	13. Project	14. Other Reference

15. Distribution statement

Unlimited

Descriptors (or keywords)

continue on separate piece of paper

Abstract

This Memorandum presents the results of a one year contract (A94c/2711), carried out by TSL Communications Ltd, which considered RSRE's SMITE secure computer system. The contract provided a peer review of the architecture proposals, characterised the essential architectural elements and formulated the security oriented top level model.